CMS Tracker FED Reduced Hardware Production Testing Setup.

Matthew Noy

25th January 2004

Version 0.0

Abstract

In order that a full complement of FED boards, plus spares, be provided in a fully functional condition for the start of the LHC data taking period, it is essential that a suitable testing procedure be planned and implemented to provide the appropriate feedback during FED production. This document discusses a proposal to implement a component testing station to be placed in the Assembly Company. This station would be aimed at diagnosing manufacture errors in areas of the board not exercised by the JTAG Boundary Scan, using a minimum amount of hardware peripheral to the FED.

Contents

1 Introduction

At the analogue front end of the FED there are a substantial number of components that cannot be checked for connectivity by the JTAG boundary scan tests. This region is densely populated, and high in complexity. Thus to speed the production process it is highly desirable that there exists an automated system for identifying faults located within this region.

For this there are two automated methods at our disposal. Applying known external signals, and looking at the response of the FED, or using the individual channel, and front end module, programmable offsets to examine how the FED is able to digitise these DC signals. The first option requires a significant amount of equipment peripheral to the FED. Thus this report looks at the possibilities, requirements and potential gains of implementing a testing system designed to extend beyond the reach of the JTAG boundary scan, and provide information about the analogue front end, without the use of externally applied signals.

2 Non-JTAGable Signal Tests

This method relies on some intrinsic assumptions.

1. Power and JTAG tests have been passed
2. Firmware loading subsystems are functional
3. The optical receivers have been individually tested and verified

With these assumptions it is possible to analyse DC offsets from the Opto-Rx, and channel individually programmable $TrimDAC^{®}$. Of course, the absence of correctly shifting offsets, or an inappropriate amount of noise, with some data analysis can point a technician to the problem, or at the very least, the correct region of the board to test more specifically what functionality is missing. Using techniques like this the following non-JTAGable components can be tested for correct functionality.

1. Opto-Rx
2. $TrimDAC^{®}$
3. EL2140 Op-Amps

4. ADC
5. LM82 (FE and BE components)
6. I^2C interface to TTCrx (JTAGable as well)

These areas are discussed in turn in more detail in the following subsections.

2.1 Analogue Opto Receiver

The Analogue Opto-Receiver (Opto-Rx) devices are 43 pin surface mount, run from the +5V supply to ground. Each Opto-Rx serves 12 optical input channels, and delivers 12 single ended electrical outputs. Even though the pitch of these pins is fairly coarse (1.27mm), it is still possible to have shorts, and badly soldered joints. There are 8 configuration pins per device, connected directly to the front FPGA, that simply take a binary level. These do the following:

- Pre-Gain signal offset - 4 bits.
- Post-Gain signal offset - 2 bits
- Capacitance network configuration - 2 bits

The Opto-Rx is a current output that drives through a load resister, and thus will provide an offset voltage, controllable by the 6 offset pins. Thus it is possible to verify the connectivity of the device, by toggling these offsets, and capturing data about the DC level of the signal. All 12 output channels are controlled identically by the control bits. It is not thought to be possible to determine by this method (i.e. automatically and without an input signal) whether the capacitance network control bits are connected correctly. More detailed information on this component can be found in the data sheet [2].

2.2 Analog Devices $TrimDAC^{®}$

The FED URD [1] requested that there be the ability to shift the DC level of each channel indivually. This is made possible with the use of the AD8802, a 12 channel, 8-bit $TrimDAC^{®}$ produced by Analog Devices [3]. Each of the 12 channels is individually programmed using a write only, 3-wire serial interface, via the Front End FPGA. This connectivity is essential for complete operation of the device. Thus it is possible to determine if the device is functioning correctly on a channel by channel basis, by setting a level and capturing some data.

2.3 EL2140 Operational Amplifiers

At the front end, the single ended electronic signal is converted to differential, with a suitable offset, by the EL2140 [4]. This is an analogue, fully differential, fixed gain line driver. Observing a correctly shifting signal will imply that this component is connected, and functioning correctly.

2.4 AD9218 Analogue to Digital Converter

All tests rely on the correct functionality of the FED ADC, the AD9218 [5]. This is a dual, 10 bit Analog Devices part, which supports two input capture ranges (1Vpp and 2Vpp), and two output data formats (offset binary and twos complement). The mode is selected by signals joined directly to the Front End FPGA, and although only 1 mode of operation is expected to be in use (1Vpp, offset binary) it is essential to know that the correct connectivity has been achieved during the assemble process. Further, the output lines are joined to the Delay FPGA, through series damping resistors of 22Ohms. These resistors come in extremely small packs, and are prone to solder bridges between adjacent signal lines. Those on the Delay FPGA side of the resistor pack are usually picked up by the JTAG boundary scan, however, those that occur on the side of the AD9218 are often missed. It is unknown to the author at present if this sort of feature will be identified by AOI.

2.5 LM82 Temperature Monitor

The purpose of this chip is to implement temperature sensing, and automated shutdown procedures for exceeded temperature limits on the FED. Instances of this chip have been placed in several locations on the FED card. It is addressable by a read-write, 2 wire serial interface [6]. Successful transactions over the SMBus should should verify both, that the device is connected correctly and is functional. Thresholds for the powerdown signal can be read back, and thus verified in that fashion, and sensible[1] (see section 2.6) values for the local and remote temperatures should indicate that the temperature sensing is functional. As a final test, the thresholds for power cutoff could be set low enough to activate the circuit, and thus tested in that manner.

[1]the exact meaning of this phrase is yet to be determined

2.6 The LHC Crate as a Measurement Instrument

The LHC crates come with a CANBus interface, that enables both the setting of crate parameters (fan speeds etc.) and the reading of crate status. Thus, an interface to the test crate would enable the following parameters to be measured[9].

- Total Power Drawn
- Total current drawn on each of the supply voltages
- Supply voltage level
- Crate temperature

Although somewhat less than ideal, since the resolution of the current measurements is somewhat low, gross problems should be identifiable, once a realistic picture of the FED power usage is built up.

Further control of the crate fan speed should allow the temperature sensors (section 2.5) to be verified in a more proactive way, since the crate fan speed could be varied whilst looking for the corresponding change in temperature being read out from the LM82s.

2.7 Other Components

Connectivity of the QDR memories at the FED back end will have been verified by the JTAG boundary scan, the data readout can be made to use these, hence verifying their internal functionality.

Since the aim of these tests is to provide addition debugging information about areas of the board that are unreachable by JTAG boundary scan, with the minimum additional hardware, it is difficult to see what additional parts of the FED can be tested without introducing more system components (section 3).

3 Requirements

3.1 Hardware

So far the hardware requirements would be:

- Linux PC
- SBS (or equivalent) PCI-VME bridge
- 9U VME crate with VME64x Backplane
- CANbus PCI Card interface and suitable cable

The status readout and control of the crate requires a CANBus PCI bridge and appropriate cable. This is available from the CERN pool [10] at a cost of 8CHF/Month at the time of writing. There is no CERN Linux support, but the kernel drivers are available at the KVaser web page [11] at no cost. Some software effort would be required (section 3.5.1).

3.2 Firmware

Firmware and software library requirements are strongly coupled, and could be met in two ways. The preliminary FED test firmware developed and the associated software libraries developed at Imperial College in April/May 2003 (by the author), or the online FED firmware and software system (currently under development and testing). Although the former option has its benefits, the author recommends the latter.

3.3 Software: Library Requirements

Several software libraries will be required to be installed on the Linux PC. These include the following:

- CERN Linux
- Drivers for PCI-VME Bridge
- Root libraries
- Hardware Access Libraries (HAL)
- FED Access Libraries
- Test Libraries (detailed in section 3.4)
- KVaser PCI-CANBus Kernel drivers [11]

Is is not expected that the XDAQ wrapper be required since no additional hardware should be required, and hence there will be no additional modules with which the board needs to communicate.

In the final system, (the author believes) CERN intends to monitor the crates using Windows, and thus so far have only done so using this OS. This implies an additional software overhead, since an interface to the Linux Kernel drivers should be written. Exactly how long this will take is dependent of the level of abstraction already provided.

3.4 Software: Test Requirements

Producing the tests aimed at a specific area of the board is the largest remaining area of work. These should include tests that look at the following components/issues on the FED:

1. Check/monitor power consumption/crate voltage levels
2. Write to, and then read from and check, all possible FED registers
3. Check Opto-Rx response for every channel
4. Check TrimDAC response for every channel
5. Use ADC modes to verify ADC control connections
6. Check LM82 response
7. Use combinations of Opto-Rx and TrimDAC offsets and check noise at all signal levels
8. Use crate fan control to fluctuate the temperature of the FED, and check against measurements

This list is not intended to be exhaustive, and is discussed in more detail below.

3.4.1 Generic Test Overview

These tests fall naturally into several categories, those associated with the monitoring of the device, and those associated with the component connectivity and correct operation.

The first set rely on accessing status registers to read the values, and measure the operation of the device (such as current drawn on each voltage level, device temperature, and internal status registers during data capture).

Those in the second set will proceed by setting a specific configuration for the component under test, then issuing a software trigger and capturing the data from the readout chain. As there will be no (Opto-Rx) input data the FED will have to be operated in scope mode, or the spy channel utilised. After data are captured over the range of values required for a specific diagnosis to be performed, the analysis required to produce that diagnosis must be done. Finally, a report card for the FED must have the diagnosis for the test appended to it.

All data should be kept, and managed in a database constructed by the test bench framework. A scheme for this is under consideration by the author at present.

3.4.2 Crate Tests

Once the available functionality of the crate is provided in an abstracted C++ API, this test should be performed first. It is possible, and may be desirable, to control the power on sequence of the crate via software. An evolving picture of the FED power usage profile should be built up during the course of production.

However since large problems with the power systems will have been identified during the specific power testing, perhaps the most powerful aspect of using this functionality will be to observe FED component temperatures as a function of crate fan speed. This test would be expected to proceed as the setting a fan speed,followed by a stabilisation time, and then readout of the temperature at all possible points on the board.

3.4.3 Register Tests

Typically this type of test should proceed as follows: All registers should be written to on the FED. The completion of serial commands should give an indication of register and serial link functionality. Read registers should then be read back and the value obtained checked against the uploaded value. This cycle should be repeated with specific test patterns that check that each bit on the FED can accept and retain a zero and one (when set to do so. . .).

3.4.4 Opto-Rx

For a default TrimDAC setting (mid-range, after a reset) the level on all 12 channels in a front end module should change the value captured. All levels allowed by the device (i.e. ignoring the bits that configure the capacitance network) should have a capture made on them, and the data saved to disk.

Comparisons of this data between channels in a given front end module, and against estimated good values should be made to try to locate possible sources of errors.

3.4.5 TrimDAC

For the default Opto-Rx offset, the TrimDAC on every channel should have the offset varied from the minimum possible to the maximum possible setting. A given device should have the data values set differently for a given capture, to validate the address decoding in the device, but there is no reason why neighbouring Front End Modules should have different treatment. (The exact scheme of achieving this is beyond the scope of this report, but some considerations for time of test, and volume of data have to be made).

Captured data of this type should be reconstructed to produce the characteristic plots. The offset profile for each channel should be fitted, and the slope, linearity, and clamping points determined.

A comparison of these points between channels, and against expected values should be made, and an estimate of the functionality of each device made.

3.4.6 LM82

The internal registers on the LM82 should be tested for write/read consistency during the Register Tests (section 3.4.3). There are, however, additional tests to verify device functionality. Where possible, the $\overline{INT}$ signal should be read back and checked against the value it should be when the local and remote $HIGH$ value is set to be lower than the measured temperature. It may also be desirable to exercise the T_CRIT set point, and verify the correct FED behaviour.

Finally, the LM82 devices on the board should be able to be used in conjunction with varying the crate fan speeds. It may be possible, with data, to find a dependence of FED temperature on crate fan speed, and ambient air temperature. The latter should be obtainable from the crate temperature sensor (although the location of this sensor appears to be undefined).

3.4.7 Advanced Tests

need a bit more time on this one...

3.5 Development Time Scales

3.5.1 Libraries

For the FED, the software library development is something that goes hand-in-hand with the firmware development. It would be possible now to implement most, if not all, of the tests described above using the current versions. The approach used by the author to the FED software libraries has been to request a shift towards a “link to” rather than “compile with” relationship. This makes for easy upgrade when necessary or desired.

Other components, such as the HAL and ROOT, may be approached from a “user” point of view, i.e. that the modules are present and functional as black boxes.

Areas in need of development are the crate readout libraries, and the Test Bench library. It should be possible to develop the crate readout libraries within a few weeks of having the necessary hardware. Development of the Test Bench library itself is currently underway, and a working first version exists [12], and an introduction can be found in appendix A. However, depending on exactly what functionality is decided to be necessary, the time required to provide it could be as much as a month or 6 weeks.

The chronology of the development of the tests themselves is discussed in section 3.5.2.

3.5.2 Tests

Development of the tests detailed in sections 3.4.2 to 3.4.7 could be expected to take something of the order 6 to 8 weeks, depending on the level of sophistication required, and the number of developers. For a system to be placed in the Assembly Company chosen, 10 weeks is a realistic estimate.

4 Summary

Production testing should be able to benefit from the additional information provided by a system like to one described above. Care should be taken to reduce the reach of effects like those seen in the last quarter of 2003, and the faster the testing feedback loop, the better this can be achieved. Requirements for a system that augments the JTAG boundary scan tests are discussed, along with the requirements. The majority of remaining work falls in the software domain, in the area of writing the tests, and checks.

References

[1] User Requirements Document for the Final FED of the CMS Silicon Tracker Version 0.51, K.W.Bell et al. Feb. 2002.

[2] Analogue Opto Receiver Data Sheet, http://cms-tk-opto.web.cern.ch/cms-tk-opto/tk/default.htm

[3] AD8802 Data Sheet, http://www.analog.com/

[4] Elantec EL2140 Data Sheet http://www.intersil.com/

[5] AD9218 Data Sheet, http://www.analog.com/

[6] The LM82 Product information, and Data Sheet, http://www.national.com/pf/LM/LM82.html

[7] The Root Homepage, http://root.cern.ch/

[8] The Hardware Access Libraries, http://cmsdoc.cern.ch/˜cschwick/software/documentation/HAL/index.html

[9] The LHC Subrack/Crate Project: Crate Specification, http://ess.web.cern.ch/ESS/crateProject/

[10] CERN Electronics Pool, http://ess.web.cern.ch/ESS/electronicsPool.htm

[11] KVaser Homepage, http://www.kvaser.se/

[12] A Test Bench for C++ Code Integration and Execution. M.Noy, January 2004.

A Introduction to A Test Bench for C++ Code Integration and Execution

A.1 Test Parameters

Firstly the Test Bench defines a test element, implemented as the class `TBSequencerElement`. Instances of this class should be able to parameterise a test, specifying the files required for the input data, where to put the output, how to execute the element etc. All of the parameters are accessible via setter and getter methods. Further there is a GUI that allows their creation in a "point and click" manner, as well as the creation and manipulation of sequences of elements (section A.2).

An element logically has 3 parts to it. The first part is an initialise function, that comes as part of the Test Bench. This simply instantiates `Fed9UDevice` and `FEDtesterEnsemble` objects with the parameters stored in the element, and makes pointers to these objects accessible to the user[2]. Secondly, there is the user Test itself, which consists of code written to perform a specific hardware test. Thirdly there is a Check function, for the purpose of performing some analysis on acquired data. This is a user defined function that must be written, and inserted (see section A.4) into the Test Bench by the same method as the test. The user has the option not to have a Check, and also to disable it if present.

Every element in a sequence must have a name. This name is used as the handle the Test Bench uses to identify the code the user wants to execute. The handle must be specified, and identified with the code in the test file.

A.2 Sequences of Test Parameters

Next the Test Bench provides a sequence of `TBSequencerElement`s in the class `Fed9UTestBenchTestSequencer`. This class contains a list of `TBSequencerElement`s and implements manipulators that allow the sequence to be customised. Elements can be added, deleted, edited, moved up or down, and loaded from, and saved to disk. Further, whole sequences can be appended. Thus the tests executed, and the order of their execution is decided at run time, in a fast and flexible manner.

[2]The test bench architecture is a general one, and is not dependant on the presence of a FED or FEDtester. It is, of course, possible to define tests that require no hardware at all.

A.3 Excution of Sequences

One (and only one) `Fed9UTestBenchTestSequencer` class instantiation is used by the `Fed9UTestBench` class which forms the user interface, and the execution core of the Test Bench. Through an instance of this class, the user can load a sequence and execute the tests associated with the elements of that sequence, one after the other, in the order that they occur. This runtime program flow control implies that the entry point, once sufficiently defined, should only need to be changed very rarely, if at all.

Execution of the sequence utilises a multi threaded approach, such that the execution may be cleanly controlled without the use of signal handlers Users can define points within the code they write for the test bench at which operations, such as pause the execution, jump forwards or backwards in the sequence, can be performed at a time when the state of the hardware resources in use is known.

Actual implementation details of the Test Bench need not be known by the user, as a result of the implementation of a PreProcessor. This is designed to automate the insertion of user written code into the Test Bench and deal with the details of registering that code with the framework, such that it may be called as and when required (see section A.4).

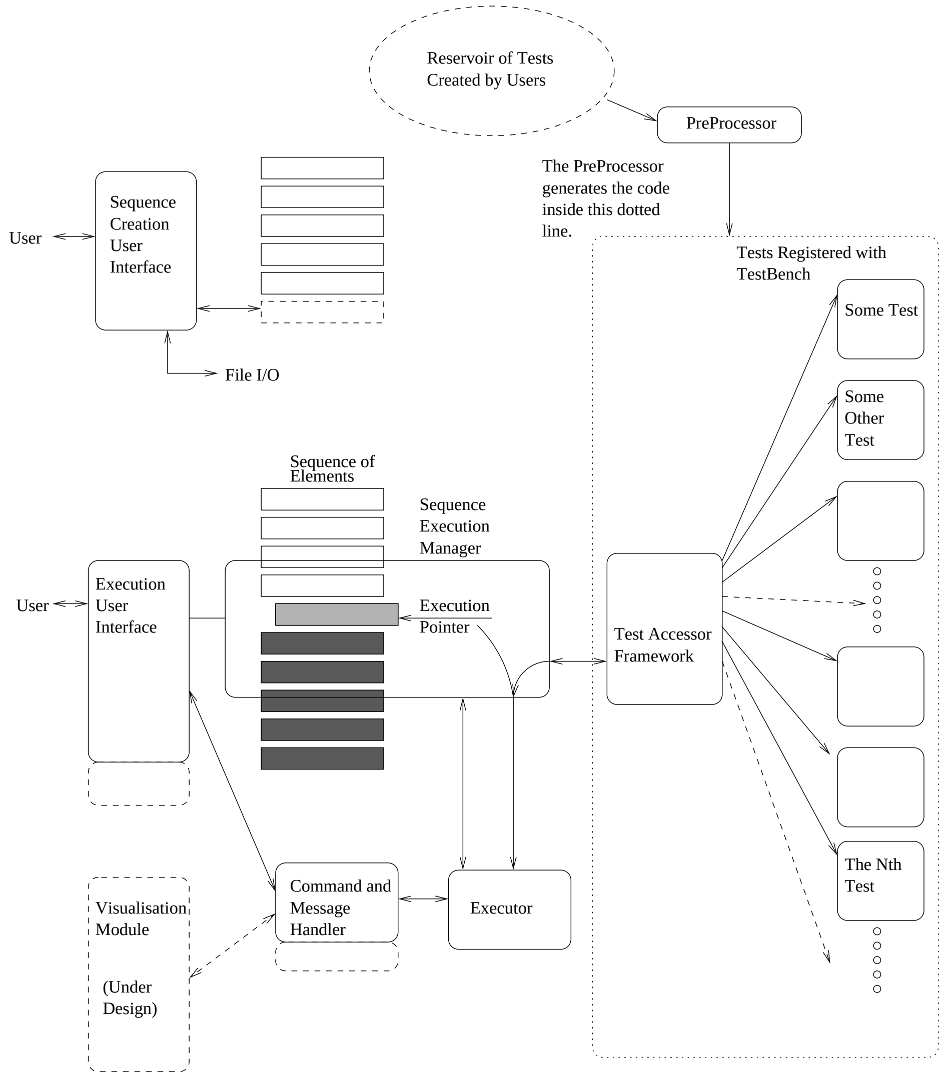
A.4 Adding Tests for Execution

In order to reduce the learning overhead, a system has been provided to abstract the details of accessing the flexibility described above. This system uses a PreProcessor mechanism where the user embeds standard C++ within special tags, and then calls the PreProcessor on that file. The PreProcessor takes the code, and embeds it in to the Test Bench, updating all of the required files while doing so. In this manner, not only can there be many "users" all writing code for the same Test Bench, the Test Bench is protected against having the execution core accidentally damaged, since this part should never be edited by somebody developing tests.

Once the PreProcessor has completed, the Test Bench can be recompiled. A Makefile that specifies that the PreProcessor must be run before the compiler and linker, makes these two steps almost indistinguishable.

Figure 1 presents a schematic of the currently implemented test bench architecture and functionality. The User Interfaces provide access to the sequence creation and manipulation features (top left). The PreProcessor parses the reeservior of user tests and constructs the Test Accessor Framework. The execution manager deals with sequence execution, and the messaging subsystem handles the user control, both interactive and programmed.

Development is expected to yield a visualisation service, and improve robustness, increase provided services, and further increase usability.



Reservoir of Tests Created by Users
PreProcessor
The PreProcessor generates the code inside this dotted line.
Tests Registered with TestBench
Some Test
Some Other Test
The Nth Test
User
Sequence Creation User Interface
File I/O
Sequence of Elements
Sequence Execution Manager
Execution Pointer
User
Execution User Interface
Test Accessor Framework
Command and Message Handler
Executor
Visualisation Module
(Under Design)

Figure 1: Schematic of Test Bench Architecture. Dashed lines represent absent or incomplete components.